Computer Engineering and Applications Vol. 6, No. 3, 2017

# Python Source Code Plagiarism Attacks in Object-Oriented Environment

Oscar Karnalim, Aldi Aldiansyah
*Maranatha Christian University*
*oscar.karnalim@it.maranatha.edu, lietafei93@gmail.com*

## ABSTRACT

Since source code plagiarism is an emerging issue on Computer Science major and Python is a new popular programming language, this paper aims to empirically enlist plagiarism attacks that might be occurred on Python source code. As our case study, our work will be focused on source code plagiarism in object-oriented environment. The result of this work is expected to become either an evaluation baseline or a prior knowledge for developing Python-targeted plagiarism detection system. Based on 280 plagiarism-suspected pairs that were extracted from four Basic Data Structure classes, four findings can be deducted. First, there are 20 distinct Python plagiarism attacks that might be occurred in object-oriented environment. Second, plagiarism attack trend on both object-oriented and procedural environment are considerably similar to each other. Third, there is no need to handle plagiarism attacks in both object-oriented and procedural environment separately. Last, plagiarism attacks in object-oriented environment is more monotonous than such attacks in procedural environment.

**Keywords:** Source Code Plagiarism, Plagiarism Attack, Python, Object-Oriented Environment, Undergraduate CS Education.

## 1. INTRODUCTION

Source code plagiarism refers to an act of reusing other people's code without acknowledging the original author beforehand [1]. It is a big concern on undergraduate Computer Science major since, on such major, most assignments are submitted electronically and such electronic representation can be easily replicated in a no time [2]. To handle such issue, several automatic plagiarism detection systems have been developed. These systems are expected to extenuate the burden of detecting such illegal behavior.

As our long-term goal, we plan to propose a Python-targeted source code plagiarism detection system for programming assignments in our university. However, since developing a solution without knowing the problem well might cause an incompatible solution, this paper will act as a prior work to map the problem (i.e. possible attacks that might be occurred on programming courses) comprehensively. It will be conducted based on 280 plagiarism-suspected pairs collected from four undergraduate classes of Basic Data Structure course. We are aware that Karnalim has done similar work on Introductory Programming course [3]. Hence, we complement his work by focusing on two new contributions. First, we will enlist plagiarism attacks in object-oriented environment. Second, we will check whether the trend of plagiarism attacks in object-oriented environment is similar to such trend in procedural environment.

## 2. RELATED WORK

According to Al-Khanjari et al [4], by excluding hybrid approach, source code plagiarism detection system can be classified into two categories, namely attribute-based and structure-based approach. On the one hand, attribute-based approach refers to an approach which relies on key properties to determine similarity. The key properties are extracted from both compared-to-be source codes and then compared using a particular similarity algorithm. To enhance the accuracy, some of them use approximate-matching algorithm that has been well-used on Information Retrieval domain [5], [6] or Machine Learning domain [7]. In such manner, detected cases are not only limited to verbatim copy but also partially-similar copy. On the other hand, structure-based approach refers to an approach which relies on source code structure to determine similarity. Source codes are converted to internal representation, such as source code token [8], [9] or low-level token [2], [10] and then compared to each other using a string similarity algorithm. According to several works [11], [12], this kind of approach outperforms attribute-based approach in terms of effectiveness toward most plagiarism cases.

When observed further toward these plagiarism detection systems, most of them were developed by viewing source code plagiarism from black-box perspective. They did not consider the detail of plagiarism characteristics before developing the system. Hence, we would argue that some of them might not be applicable on real environment. This paper aims to fill such gap by providing plagiarism characteristics. Such characteristics will be generated by enlisting possible plagiarism attacks that might be occurred on programming courses. For our case study, we will focus on Python plagiarism attacks in object-oriented environment. It is important to note that this work complements Karnalim's work [3] by focusing on two new contributions. First, our work will be focused on object-oriented environment, an environment that has not been discussed by Karnalim's work. Second, our work will check whether the trend of plagiarism attacks in object-oriented environment is similar to such trend in procedural environment.

Besides providing a prior knowledge for developing a Python-targeted source code plagiarism detection system, there are two other benefits that can be extracted from this work. First, listed Python plagiarism attacks can be used as evaluation metrics for evaluating already-developed plagiarism detection system. Second, plagiarism attack trend can be used as a part of prior consideration before developing a Python-targeted source code plagiarism detection system. If the trend of plagiarism attacks in object-oriented environment is similar to such trend in procedural environment, then it is unnecessary to handle plagiarism attacks in both environments separately. Otherwise, plagiarism attacks should be handled separately based on their environment.

## 3. RESULTS AND DISCUSSION

In general, there are two research questions proposed in this work. First, what kinds of plagiarism attacks are occurred on object-oriented environment? Second, is plagiarism attack trend in object-oriented environment similar to such trend in procedural environment? Both questions will be answered by conducting our four-fold proposed methodology. The first research question will be answered on the third phase while the second one will be answered on the fourth phase.


ISSN: 2252-4274 (Print)
ISSN: 2252-5459 (Online)

Our proposed methodology consists of four phases that should be executed in sequential manner. These phases are raw data collection, plagiarism-suspected pair filtering, manual classification of plagiarism attacks, and trend analysis.

First, raw data collection is responsible to collect student source codes for our dataset. This phase is implemented by collecting all submitted source code projects from four undergraduate classes of Basic Data Structure (BDS) course. Such course was held in the even semester of 2016/2017 academic year where each student is required to submit 14 source code projects for the whole semester. In terms of involved built-in functions and syntaxes, this course extends function and syntax set from Introductory Programming course [3] by incorporating standard object-oriented concept namely class, constructor, method, and attribute. It is important to note that BDS course is selected as our case study instead of Object-Oriented Programming (OOP) course since we want to check plagiarism attack trend on a course that utilize object-oriented environment as its supplementary topic, not the main one. That is why we use the term Object-Oriented Environment instead of Object-Oriented Programming as our terminology.

Second, plagiarism-suspected pair filtering is responsible to select the most representative plagiarism-suspected pairs. This phase is implemented by generating plagiarism-suspected pairs in pairwise manner for each assignment per class. To ensure that such pairs are plagiarism-suspected pairs, we will only take pairs that satisfy two rules. First, both projects should be graded with a score higher or equal to 80 of 100. Such rule is applied to ensure that both projects share the same goal (i.e. solving the problem correctly), as it is known that two source codes with different goals cannot be considered as a plagiarism pair. Second, similarity degree between both projects should be more or equal to average similarity threshold on such assignment. Such rule is applied to ensure that both projects share a considerably high similarity, as it is known that high similarity is one of the key factors to determine plagiarism. Similarity measurement is implemented as in Karnalim's work [3]. Both projects will be converted to token sequences and compared to each other using Rabin-Karp Greedy String Tiling algorithm. The only difference between his work and ours is that, in our work, since a project may contain more than one source code file, token sequence for each project will be formed as the concatenation of token sequences resulted from source code files on that project.

Since the authors of this work are required to classify automatically-listed plagiarism attacks manually at the third phase, not all pairs are fed to the third phase. Instead, for each assignment, we will take 5 random pairs per class. We prefer to select such pairs randomly rather than only taking pairs with the highest similarity degree since some plagiarism attacks might not be found on top pairs due to their significant modification. As a result, there are 280 plagiarism-suspected pairs. These pairs are generated by taking 5 random pairs per class per assignment where there are 4 classes and 14 assignments.

Third, manual classification of plagiarism attacks is responsible to empirically enlist plagiarism attacks found on filtered pairs. It is the only phase that will be conducted manually by the authors. In terms of responsibility, the second author is responsible to enlist initial attack list while the first author is responsible to evaluate such list and perform some corrections if necessary.

Last, trend analysis is responsible to check whether the trend of resulted plagiarism attack list is similar to the trend of plagiarism attack list in procedural environment. This phase will be conducted by comparing the frequency distribution of Karnalim's and our plagiarism attack list using Pearson correlation [13].

## 4. RESULT AND DISCUSSION

### 4.1 What kinds of plagiarism attacks are occurred on object-oriented environment?

According to our proposed methodology, 20 distinctive plagiarism attacks are extracted from 280 plagiarism-suspected pairs. The detail and occurrence frequency of these attacks can be seen on Table 1, sorted in descending order of occurrence frequency. Similar with attacks listed on Karnalim's work [3], each attack works in reversible fashion. For instance, if an attack is focused on incorporating a dummy method, then removing dummy method is also considered as that attack.

TABLE 1.
Plagiarism Attack List

| Attack Type | Occurrence Frequency (Pairs) |
|---|---|
| Modify comment | 237 |
| Modify local variable name | 220 |
| Modify whitespace | 91 |
| Incorporate dummy instructions without changing the decision logic | 45 |
| Modify method name | 37 |
| Incorporate logical expression that can be replaced with boolean constant | 35 |
| Rearrange method declaration | 30 |
| Rearrange loosely-coupled instructions on similar scope | 29 |
| Break down API-based instruction to several more-specific API-based instructions | 24 |
| Modify class name | 23 |
| Rearrange branching statements based on its condition validation sequence | 15 |
| Reuse declared variables for other processes | 14 |
| Exchange API-based instruction with other API-based instruction that yield similar functionality for particular circumstance | 11 |
| Encapsulate a particular task as a void method with the use of global variables | 7 |
| Encapsulate a particular task as a void method without the use of global variables | 4 |
| Incorporate dummy methods | 4 |
| Change loop type | 2 |
| Change loop boundary | 2 |
| Change incorporated algorithm with another algorithm which shares similar goal | 2 |
| Assign different default value to a variable | 1 |


ISSN: 2252-4274 (Print)
ISSN: 2252-5459 (Online)

### 4.2 Is plagiarism attack trend in object-oriented environment similar with such trend in procedural environment?

The occurrence frequency distribution of listed plagiarism attacks from Karnalim's work [3] and ours can be seen on Figure 1. Vertical axis represents normalized occurrence frequency degree in percentage. It is calculated by dividing the frequency with the total number of involved pairs for each dataset. On the contrary, horizontal axis represents merged plagiarism attack set from Karnalim's and our work. Since the number of distinct plagiarism attacks found in our work is lower than Karnalim's work, we map our attacks to his list and only leave the attack as it is if it is not listed on Karnalim's work. For convenience, on Figure 1, each Karnalim's listed attack will be prefixed with KP and followed by its unique ID where the detail for such ID can be seen on his work [3]. It is important to note that, since our attack list put more details on comment, whitespace, and identifier name modification, some attacks are merged to fit Karnalim's attack scope. Such merging mechanism is conducted by recounting such frequency toward our dataset based on Karnalim's attack scope.

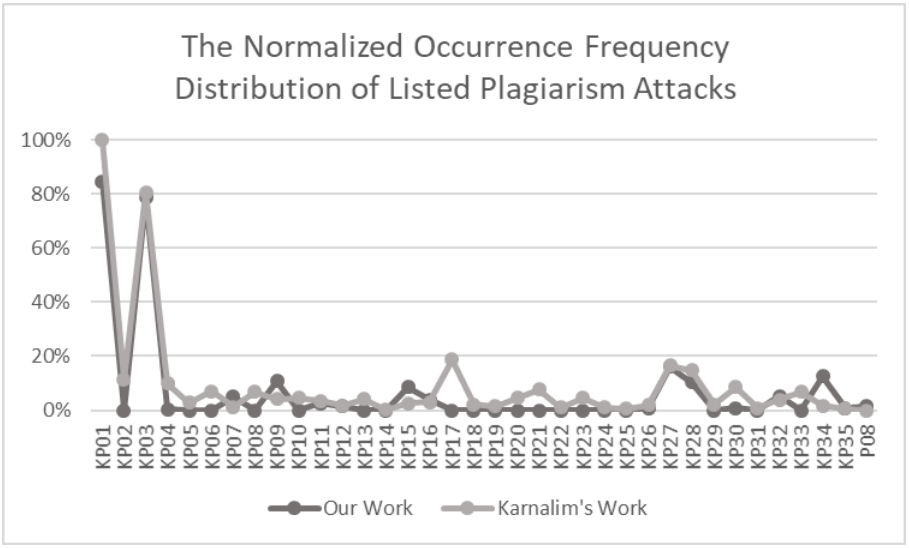


FIGURE 1. The Normalized Occurrence Frequency Distribution of Listed Plagiarism Attacks

When correlation between both trends is measured using Pearson correlation, it is clear that both trends share high similarity. It generates 0.961 while the maximum similarity score for such correlation is 1. Hence, it can be stated that plagiarism attack trend in object-oriented environment is similar with such trend in procedural environment. Such finding is also strengthened by the fact that both trends are visually similar to each other on Figure 1.

## 5. CONCLUSION AND FUTURE WORK

In this paper, we have enlisted Python plagiarism attacks from four classes of Basic Data Structure course. For each class, we have taken all programming assignment that were submitted during the even semester of 2016/2017 academic year. Such assignments are expected to represent Python source codes on object-oriented environment. In addition to providing plagiarism attack list, there are three additional findings that can be deducted. First, plagiarism attack trend in both object-oriented and procedural environment are considerably similar to each other. Second, there is no need to handle plagiarism attacks on both environments separately since the frequency distribution of Python plagiarism attacks in both environments are considerably similar. Last, plagiarism attacks in object-oriented environment is more monotonous than such attacks in procedural environment since the number of distinct plagiarism attacks on our work is significantly lower than such number on Karnalim's work.

For future work, we plan to use this work, along with Karnalim's work [3], as a baseline to develop a Python-targeted plagiarism detection system. Moreover, we also plan to enlist plagiarism attacks on other Python-related courses to strengthen and enrich our findings.

## REFERENCES

[1] H. A. Maurer, F. Kappe, and B. Zaka, "Plagiarism-a survey.," *Journal of Universal Computer Science*, vol. 12, no. 8, pp. 1050–1084, 2006.

[2] F. S. Rabbani and O. Karnalim, "Detecting Source Code Plagiarism on .NET Programming Languages using Low-level Representation and Adaptive Local Alignment," *Journal of Information and Organizational Sciences*, vol. 41, no. 1, pp. 105–123, Jun. 2017.

[3] O. Karnalim, "Python Source Code Plagiarism Attacks on Introductory Programming Course Assignments," *Themes in Science and Technology Education*, vol. 10, no. 1, 2017. In Press

[4] Z. A. Al-Khanjari, J. A. Fiaidhi, R. A. Al-Hinai, and N. S. Kutti, "PlagDetect: a Java programming plagiarism detection tool," *ACM Inroads*, vol. 1, no. 4, p. 66, Dec. 2010.

[5] D. Ganguly, G. J. F. Jones, A. Ramírez-de-la-Cruz, G. Ramírez-de-la-Rosa, and E. Villatoro-Tello, "Retrieving and classifying instances of source code plagiarism," *Information Retrieval Journal*, pp. 1–23, Sep. 2017.

[6] M. Mozgovoy, S. Karakovskiyz, and V. Klyuev, "Fast and reliable plagiarism detection system," in *2007 37th annual frontiers in education conference - global engineering: knowledge without borders, opportunities without passports*, 2007, p. S4H–11–S4H–14.

[7] A. Jadalla and A. Elnagar, "PDE4Java: Plagiarism Detection Engine for Java source code: a clustering approach," *International Journal of Business Intelligence and Data Mining*, vol. 3, no. 2, pp. 121–135, 2008.

[8] J.-S. Lim, J.-H. Ji, H.-G. Cho, and G. Woo, "Plagiarism detection among source codes using adaptive local alignment of keywords," in *Proceedings of*


ISSN: 2252-4274 (Print)
ISSN: 2252-5459 (Online)

*the 5th International Confernece on Ubiquitous Information Management and Communication - ICUIMC '11*, 2011, p. 1.

[9] Y.-C. Jhi, X. Jia, X. Wang, S. Zhu, P. Liu, and D. Wu, "Program Characterization Using Runtime Values and Its Application to Software Plagiarism Detection," *IEEE Transactions on Software Engineering*, vol. 41, no. 9, pp. 925–943, Sep. 2015.

[10] O. Karnalim, "A Low-Level Structure-based Approach for Detecting Source Code Plagiarism," *IAENG International Journal of Computer Science*, vol. 44, no. 4, 2017. In Press

[11] Z. Duric and D. Gasevic, "A Source Code Similarity System for Plagiarism Detection," *The Computer Journal*, vol. 56, no. 1, pp. 70–86, Jan. 2013.

[12] L. Prechelt, G. Malpohl, and M. Philippsen, "Finding Plagiarisms among a Set of Programs with JPlag," *Journal of Universal Computer Science*, vol. 8, no. 11, pp. 1016–1038, 2002.

[13] K. Pearson, "Note on Regression and Inheritance in the Case of Two Parents," *Proceedings of the Royal Society of London*, vol. 58. Royal Society, pp. 240–242.

ISSN: 2252-4274 (Print)
ISSN: 2252-5459 (Online)